Abstract

This paper describes the Karta-CASA project, an attempt to design an agent framework that provides a flexible, extensible and easy-to-use base for constructing various single-user and distributed groupware visualization and concept mapping applications. The structure and the development process behind Karta-CASA are detailed, and several novel aspects of the system are described. Details of several applications based on the framework are provided. Further, this paper suggests several research directions for extending the Karta-CASA system into areas that have not yet been explored by other agent visualization systems.

The Karta-CASA framework: Concept mapping and multi-agent systems

Vladimir Sedach, Eunice Lim, Kurtis Fraser and Rob Kremer
vsedach|lime|fraserka|kremer@cpsc.ucalgary.ca
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada

1 Introduction

There exist many methods of visually presenting and manipulating information. Concept mapping, defined by Kremer [11] as "an intuitive visual knowledge representation technique," is a general method applicable to visualizing a wide variety of information sources. Multi-agent systems (MAS) [9] can be used to construct both single-user and distributed groupware [4] applications. The synthesis of these two ideas is the concept behind the Karta-CASA framework. The Karta-CASA framework project is an attempt to create a flexible, easy-to-use base for constructing either single-user or distributed groupware concept mapping, visualization, and graph drawing applications that can potentially exploit the unique capabilities of multi-agent systems. The Karta-CASA framework is based on the CASA (Cooperative Agent System Architecture) agent architecture, described in [13], and the Karta concept mapping tool, described in Section 2.

2 The Karta Concept Mapping Tool

Karta is an extensible tool allowing users to create and manipulate concept maps. Karta is written in Java, and is both a complete stand-alone application and a library that can be customized and embedded within any Java application that requires concept mapping or graph drawing capabilities. Integration with the embedding system is facilitated by both procedural interfaces and an architecture based on a variation of the command design pattern [6]. The visual appearance of a concept map is controlled by a novel system of hierarchical style (visual appearance property) "sheets." Karta includes a novel method for dealing with edge congestion [3] based on the EdgeLens system [25].

3 The Karta-CASA Framework

3.1 Introduction

The Karta-CASA project is an agent framework in CASA that embeds Karta in order to provide a flexible, extensible and easy-to-use base for constructing various single-user and groupware visualization and graph drawing applications. A major design goal of Karta-CASA is to take advantage of the capabilities of CASA specifically and multi-agent systems in general. Since in CASA inter-agent communication takes place using messages in the KQML [5] or XML [1] standards, it is possible for other agent frameworks or even non-agent networked systems to use the Karta-CASA framework. Several applications with varied scope and purpose have been constructed using the Karta-CASA framework, details of which are provided in Section 4. Another goal of Karta-CASA was to demonstrate that multi-agent systems in general, and CASA in particular, are a suitable and even desirable platform for building groupware and distributed soft real-time applications. The existing CASA facilities and inherent flexibility in creating new agent types simplified the construction of the framework considerably, particularly those parts of it dealing with networking and communications.

3.2 The Karta-CASA Architecture

The CASA architecture [13] is a fundamentally very flexible agent communications framework that does not impose or favor any particular application architectural style. This made the question of how to structure the Karta-CASA system an open ended one. Several approaches were considered, and from these, two were evaluated: a replicated or peer-to-peer system architecture [15, 8, 17], and a client-server architecture. Upon further progress in the design and exploration of the usage scenarios, it was found that a replicated architecture would offer few benefits for the intended use of Karta-CASA over a client-server approach while being more complicated to implement. In consideration of this conclusion, the client-server architecture was chosen as the basis for Karta-CASA.

The next step in the design involved choosing the particulars of the client-server architecture implementation. This phase involved several iterations throughout the design and implementation stages of the project. The final design blueprint can be seen in Figure 1. Note that while the terminology used and some of the aspects of the architecture resemble that of the Model-View-Controller [18], there are important differences. In the Karta-CASA architecture, the Supervisor agent acts as the user interface, embedding an instance of Karta in its own address space. The Supervisor and its Karta instance communicate synchronously using their respective procedural interfaces. The role of the Supervisor is to take any commands generated by the input to its Karta instance and send them to the View agent to which it is subscribed to (more on this below) for consideration. When a View agent receives a command from one of its Supervisor subscribers, it determines whether that command may affect

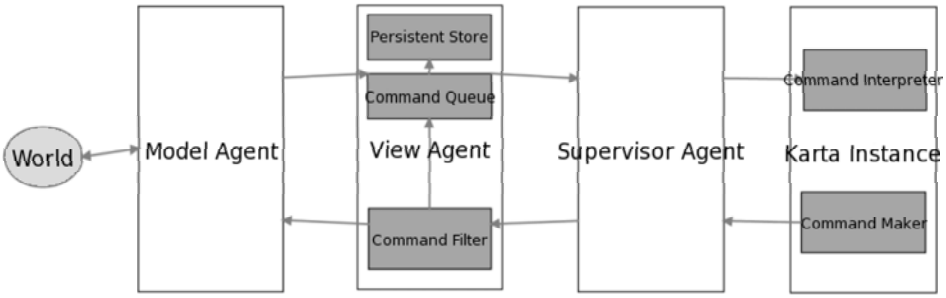

Fig. 1: A concept map of the Karta-CASA architecture.

the underlying model of the graph or not using the decision procedure of the Command Filter interface. If the command could affect the Model, it is passed on to the Model agent to which the View agent is subscribed to, and if not, the command is added to a queue to be sent to all the subscribing Supervisor agents. When a Supervisor agent receives a command from the View agent to which it is subscribed, it executes that command unconditionally, which depending on the command may produce a change to the graph on the screen. The Model agent receives commands from View subscribers, and either rejects them, or acts on them to affect the underlying model represented by the View agents. When the underlying model changes, it sends a command notifying all the View agent subscribers of the fact, which in turn send commands to all of their Supervisor agent subscribers. In addition, the Model agent keeps a persistent copy of the state of each View agent and a list of all the model changes that have occurred since a particular View canceled its subscription, enabling Karta-CASA application state to persist across work sessions.

Communication in Karta-CASA is accomplished by sending messages containing Karta commands, and occurs bi-directionally between the Model-View and View-Supervisor layers. Since one of the intended uses of Karta-CASA is building groupware applications, sequence consistency [15] was selected as a basic requirement. This is accomplished by having a subscription model, where several Supervisor agents may subscribe to a single View agent, and several View agents may subscribe to a single Model agent, the "publisher" sending sequentially numbered commands to its subscribers. The Model, View and especially the Supervisor agents lack any sophisticated reasoning facilities, mostly responding to messages as they come in, so they can all be classified as reactive agents [2, 26]. In hindsight, it is interesting to note how much similarity the Karta-CASA architecture also bears to Brooks' idea of a layered agent architecture [2]. Karta-CASA is made up of layers of asynchronously communicating agents. Information is passed from the Supervisor up to the View and then to the Model agent, each agent layer being able to subsume the role of lower-level agents by modifying or suppressing their messages, and each successive layer interpreting more of the lower layers' information in its own higher-level

context, the Supervisor agent sending raw graph commands, the View agent filtering these to Model-relevant commands and the Model agent responding to the information passed to it by the View by modifying the world or notifying the view of changes in the world as appropriate.

It is important to note that the architecture presented here is itself a pattern. From the beginning, the requirements focused on generality as a major goal of Karta-CASA. The result is that the final architecture can be customized to work with some parts omitted or expanded with additional functionality to suit a variety of applications. This generality has been made use of in existing Karta-CASA applications, details of which are provided below in Section 4.

3.3 Karta-CASA Implementation

The Karta-CASA system is implemented as three base classes and an interface: the Supervisor, View and Model agent classes, each of which extends the basic agent class of CASA, and the Command Filter interface. Applications based on Karta-CASA are built by inheriting from and extending the Supervisor, View and Model base classes and implementing the application-specific logic behind the Command Filter interface. Note that it is not necessary to implement all three classes in all Karta-CASA applications. The Model agent may be omitted for those applications that do not need to manipulate a shared underlying model. For example this is the case in the Groupware Graph Drawing and Ontology Editor applications, detailed below, where the Groupware Graph Drawing application does not have an underlying model, and the model of the Ontology Editor is not shared and its interface simple enough that integrating the model manipulation capabilities right into the extended View agent is simpler and more robust than constructing a separate Model agent. The authors believe the capability to omit a component of the framework is just as important as the features that framework provides, and the inherently loose coupling enabled and encouraged by agent-based systems is a great benefit to providing this capability.

4 Karta-CASA Applications

4.1 Groupware Concept Mapping

The Model, View and Supervisor base classes include all the functionality needed to build a Karta-CASA application (see Figure 2). Somewhat unexpectedly, this functionality was also enough to enable a useful multi-user concept mapping [12] application, where several users could edit a single concept map at the same time. This functionality was used as an early stage proof-of-concept of the Karta-CASA design, lending invaluable input to the architecture revision process, and was later dubbed the Groupware Concept Mapping application. The Groupware Concept Mapping program itself compromises nothing more than the base classes for View and Supervisor agents, that when instantiated enable several Supervisor agents to subscribe to a single View agent, the aggregate

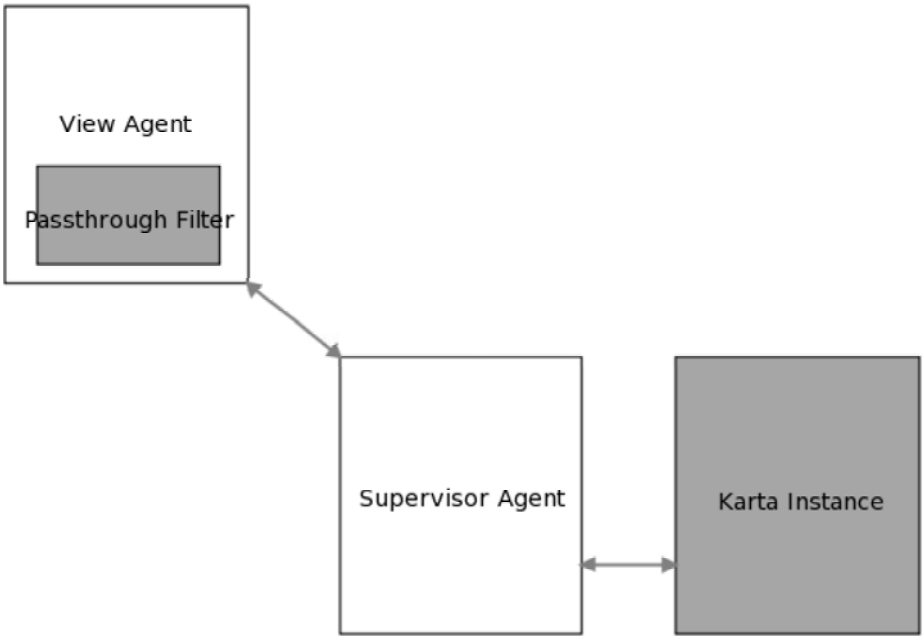

Fig. 2: A concept map of the Groupware Concept Mapping application. As the name suggests, the Passthrough Filter accepts all commands.

system acting as a groupware application for drawing concept maps, including all of the functionality of stand-alone Karta.

4.2 MASExplorer

Early on in the Karta-CASA project, the idea of having an application to visualize a multi-agent system as a concept map became a goal of the project. The initial idea of the system was to display agents and membership relationships between them to visualize the structure and relationships of a MAS in a straightforward way. In contrast to other work on MAS visualization [22, 20, 19] no attempt is made to visualize the inner workings of individual agents or to visualize the temporal changes in a MAS, which the authors thought could be more simply visualized using the stepper interface that is part of the advanced debugging facilities built into CASA. The resulting system, dubbed the MASExplorer, can now be used to examine the structure and memberships of an arbitrary MAS (see Figure 3). Although currently there is no provision for doing so, MASExplorer can be adapted to visualize a task-specific MAS with structures and graphics appropriate to that MAS model [24]. Other useful extensions to MASExplorer are the visualization of social commitments and graphic manipulation of multi-agent systems. A more in-depth discussion is provided in

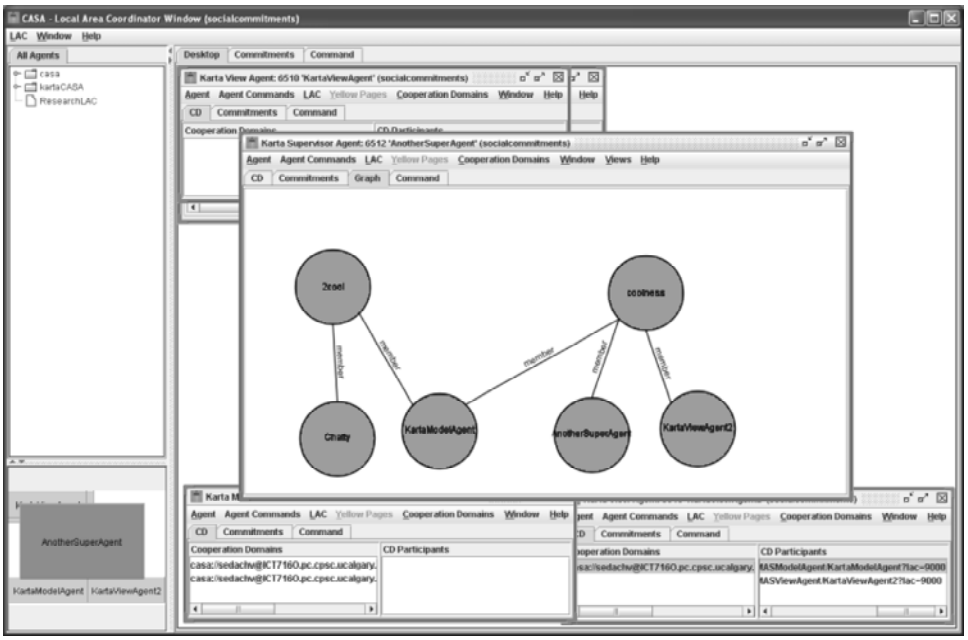

Fig. 3: A typical MASExplorer session.

Section 5.

The architecture of MASExplorer can be seen in Figure 4. Only the Model and View agents are extended with specific functionality for the MASExplorer application - the base Supervisor agent class has enough capabilities to enable it to act as an effective GUI for the application. The Model agent is responsible for gathering all information about the MAS, passing it on to the subscribed View agents to interpret and represent with View-specific options and graphics to the subscribers of a View. The built-in groupware capabilities of the View agent are leveraged to provide the option of having several users interact with a specific visualization simultaneously.

4.3 Ontology Editor

Each agent in the CASA architecture includes a performative type hierarchy (subsumption lattice), used to reason about received messages (see [10]), from here on referred to as an ontology. The ontology itself is structured as a directed acyclic graph, the content of which is a representation of knowledge, making the ontology a type of computational concept map [11] that can be conveniently visualized and graphically manipulated. The Ontology Editor application was created to do precisely that. Upon receiving a request from the user to edit the agent's ontology, a View and Supervisor agents start up. The View agent converts the ontology of the agent whose ontology is being edited into an internal

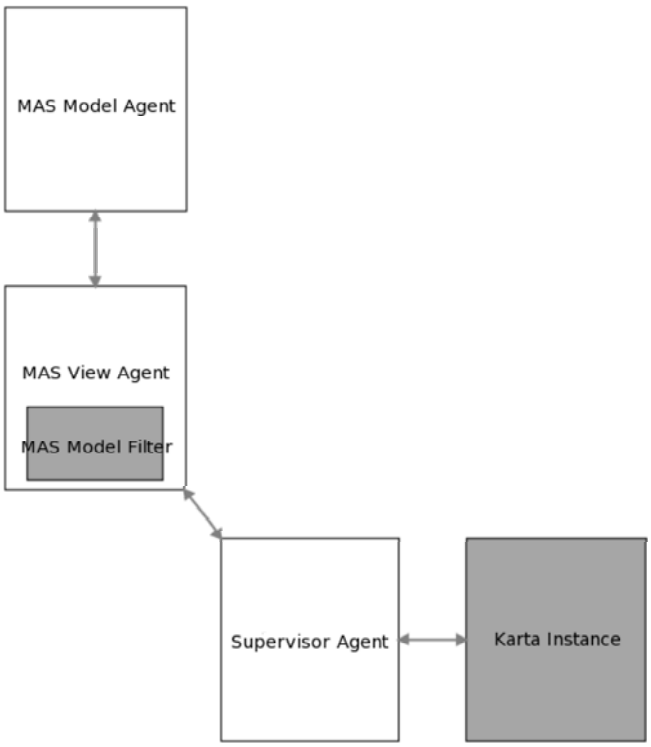

Fig. 4: A concept map of the MASExplorer architecture.

representation suitable for manipulation, then constructs a graph from its internal representation, laid out according to a graph layout algorithm adapted from Paulisch and Tichy [16] and Sugiyama [21], and then sends commands to the Supervisor to draw that graph. See Figure 5 for an example of the final result. The user can then proceed to view and manipulate the graph representation of the ontology using intuitive menu commands.

An interesting aspect of the Ontology Editor is how the Karta-CASA architecture is employed (see Figure 6). Here, the View agent is run in the same address space as the agent whose ontology is about to be edited, and communicates with that agent using the procedural interfaces defined by CASA. This means that no additional code had to be added to any of the existing CASA agents - only the separate GUI class had to be modified to add the Ontology Editor as a menu option if the Ontology Editor classes were present, via reflection [7]. The Supervisor agent is then run as it normally would be, and all the Supervisor-View agent interaction happens normally by sending messages. This means that the Ontology Editor can be used to edit the ontologies of agents on remote machines provided that the View agent class has been loaded on the remote JVM, regardless of whether those machines have GUI capabilities or not.

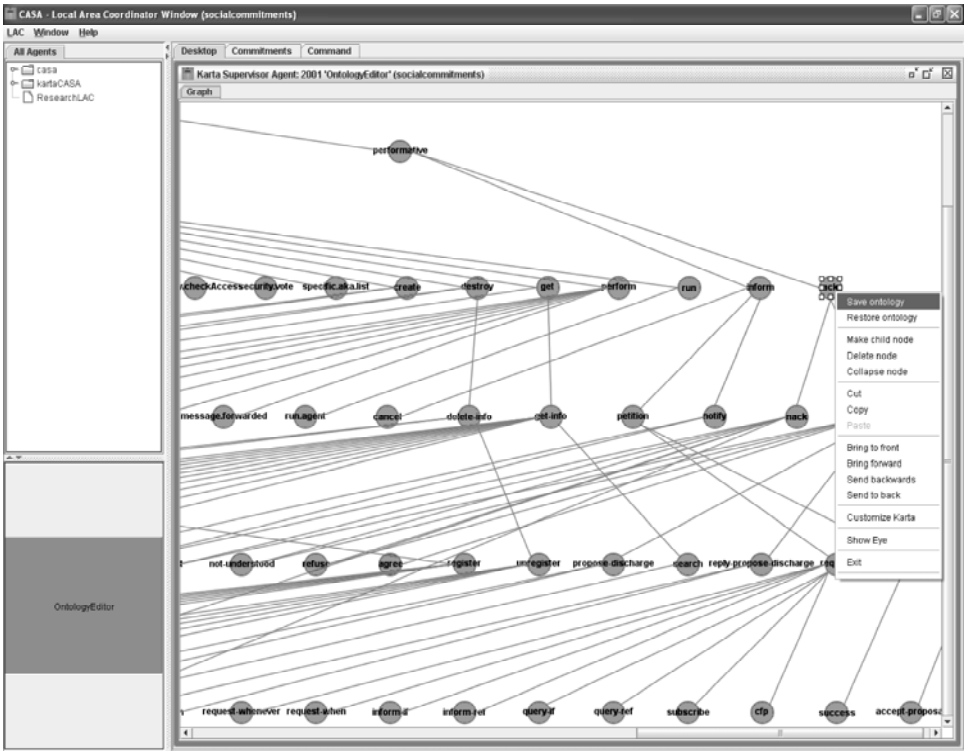

Fig. 5: A typical Ontology Editor session.

By adding some rudimentary message handling code to the agents themselves, it is possible to decouple the View agent from the agent whose ontology is being edited entirely.

5 Future research directions

One of the aspects of the CASA system is that besides sending messages to cause agents to perform actions, there also exist means for direct agent manipulation through a command-line language or a graphical interface. One of the concepts for MASExplorer was the possibility to use it as another means of directly manipulating agents. A human user would be able to request services from a MAS by finding an agent that offers those services, based on visual cues, and then request those services from that agent via menu items or direct manipulation such as dragging the representation of an agent from one place to another (for example to direct an agent controlling a robot to go from one room to another).

Another proposed extension relating to the example given above is visualizing a MAS as the model it is supposed to represent. A CASA-based MAS has certain intrinsic relationships between the agents, such as membership in

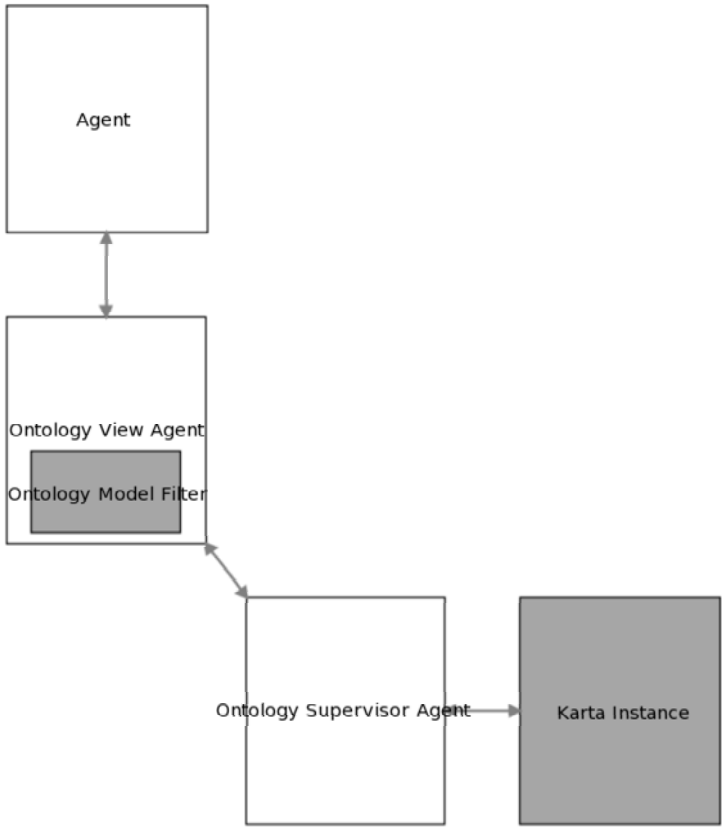

Fig. 6: A concept map of the Ontology Editor architecture. Here, Agent refers to the agent whose ontology is being edited.

cooperation domains [13], that the current version of MASExplorer is designed to visualize. In addition to these properties, a MAS is frequently used to model some external system, for example, a team of robots navigating a building. MASExplorer can be extended to visualize this model. Using the above example, a blueprint of the building can be imported and the agents controlling the robots each placed in the part of the blueprint corresponding to the robot's physical location. The proposed extension can in principle be used to make a 2-dimensional visualization of any kind of MAS model.

A third proposed extension to MASExplorer involves the visualization of social commitments [10]. This extension will be useful for both understanding the workings and interactions and for the debugging of an arbitrary MAS. The basic idea is to visualize an individual commitment as an arc pointing between the debtor and the creditor of the commitment. More research and experimentation will be required to visualize in a coherent way commitment-based conversations

[10] between agents.

A practical extension that will enhance the usefulness of Karta-CASA for constructing groupware applications is a framework for change awareness [23, 14]. The capabilities of the change-awareness framework should encompass the range of currently known techniques, from identifying what parts of the concept map are being edited by whom, to sophisticated revision control systems with intelligent branching and merging features.

6 Conclusions

This paper has introduced the Karta-CASA project, an agent framework for constructing various single-user and groupware visualization and graph drawing applications. The structure and some of the thinking behind the development process were detailed and justified, and several novel aspects of the system were described. Several applications constructed using Karta-CASA were described, with attention paid to how the framework is utilized. Further, this paper has suggested several research directions for extending the Karta-CASA system into areas that have not yet been explored by other MAS visualization systems.

References

[1] *Extensible Markup Language (XML)*, August 2005, http://www.w3.org/XML/.

[2] R. A. Brooks, *A robust layered control system for a mobile robot*, IEEE Journal of Robotics and Automation **2** (1986), no. 1, 14–23.

[3] M. S. Carpendale and X. Rong, *Examining edge congestion*, CHI '01 Extended Abstracts on Human Factors in Computing Systems (New York), ACM Press, 2001, pp. 115–116.

[4] Clarence Ellis and Jacques Wainer, *Groupware and computer supported cooperative work*, Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence (Gerhard Weiss, ed.), MIT Press, 2000, pp. 425–457.

[5] Tim Finin, Yannis Labrou, and James Mayfield, *KQML As An Agent Communication Language*, Software Agents (Jeffrey M. Bradshaw, ed.), MIT Press, 1997, pp. 291–316.

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design patterns*, pp. 233–242, Addison-Wesley Professional, 1995.

[7] Dale Green, *The Reflection API*, 2005, http://java.sun.com/docs/books/tutorial/reflect/index.html .

[8] Saul Greenberg and David Marwood, *Real time groupware as a distributed system: concurrency control and it's effects on the interface*, CSCW '94,

Proceedings of the Conference on Computer-Supported Cooperative Work, ACM Press, 1994, pp. 207–217.

[9] Michael N. Huhns and Larry M. Stephens, *Multiagent systems and societies of agents*, Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence (Gerhard Weiss, ed.), MIT Press, 2000, pp. 79–120.

[10] R. Kremer and R. Flores, *Using a performative subsumption lattice to support commitment-based conversations*, Proceedings of the Forth International Joint Conference on Autonomou Agents and MuliAgent Systems (New York), ACM Press, 2005.

[11] Rob Kremer, *Concept mapping: Informal to formal*, Proceedings of the Third International Conference on Conceptual Structures, Knowledge Acquisition Using Conceptual Graphs Theory Workshop, 1994, pp. 152–167.

[12] Rob Kremer and Brian R. Gaines, *Groupware concept mapping techniques*, Proceedings SIGDOC'94: ACM 12th Annual International Conference on Systems Documentation, Banff, Canada (New York), ACM Press, 1994, pp. 156–165.

[13] Robert C. Kremer, Roberto A. Flores, and Chad La Fournie, *A performative type hierarchy and other interesting considerations in the design of the casa agent architecture*, Workshop on Agent Communication Languages 2003 (Berlin), Springer-Verlag, 2003, pp. 59–74.

[14] L. McCaffrey, *Representing change in persistent groupware environments*, Grouplab report, University of Calgary, January 1998.

[15] Manoj Misra and Isi Mitrani, *On the propagation of updates in distributed replicated systems*, Performance Evaluation **35** (1999), 131–144.

[16] F. N. Paulisch and W. F. Tichy, *Edge: an extendable graph editor*, Software - Practice and Experience **20** (1990), 63–88.

[17] David P. Reed, *Naming and synchronization in a decentralized computer system*, Ph.D. thesis, Massachusetts Institute of Technology, 1978, also available as MIT LCS Technical Report 205.

[18] Trygve M. H. Reenskaug, *The original MVC XEROX PARC 1978-79*, http://heim.ifi.uio.no/ trygver/themes/mvc/mvc-index.html, September 2005.

[19] David Rehor, David Kadlecek, Pavel Slavik, and Pavel Nahodil, *Vat - a new approach to multi-agent systems visualization*, The 3rd IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP 2003) (Calgary, AB, Canada), ACTA Press, 2003.

[20] Michael Schroeder and Penny Noy, *Multi-agent visualisation based on multivariate data*, AGENTS '01: Proceedings of the fifth international conference on Autonomous agents (New York, NY, USA), ACM Press, 2001, pp. 85–91.

[21] Kozo Sugiyama, *Graph drawing and applications for software and knowledge engineers*, World Scientific, 2002.

[22] Pedro Szekely, Craig Milo Rogers, and Martin Frank, *Interfaces for understanding multi-agent behavior*, IUI '01: Proceedings of the 6th international conference on Intelligent user interfaces (New York, NY, USA), ACM Press, 2001, pp. 161–166.

[23] James Tam and Saul Greenberg, *A framework for asynchronous change awareness in collaboratively-constructed documents*, Proceeding of the International Workshop on Groupware (CRIWG 2004), Springer, 2004, pp. 67–83.

[24] Jeff Weston, *Visualization for multi-agent systems*, September 2005, http://mas.pf.itd.nrl.navy.mil/visualization.html.

[25] N. Wong, S. Carpendale, and S. Greenberg, *Edgelens: An interactive method for managing edge congestion in graphs*, Proceedings of IEEE Symposium on Information Visualization, IEEE Press, 2003.

[26] Michael Wooldridge, *An introduction to multiagent systems*, pp. 89–104, John Wiley & Sons, 2002.